Paul Singman
12/6/2015
Udacity MLE Nanodegreee Project #1

Predicting Boston Housing Prices

The goal of this project is to build a machine learning model to best predict the value of a given house, with the goal of calculating the best price for clients to sell their houses at.

Data Exploration

Here are some summary statistics on the dataset used to train the model:

- Size of data (number of houses): 506
- Number of features: 13
- Minimum housing price: 5.0
- Maximum housing price: 50.0
- Mean housing price: 22.53
- Median housing price: 21.2
- Standard deviation of housing prices: 9.19

Performance Metric

For the error metric, I chose Mean Squared Error (MSE) since it is an appropriate metric for a regression problem. Compared to Mean Absolute Error (MAE), this will make the mode more sensitive to large errors. In this case, I believe this is desired. For example, being off by $50 in a housing price prediction is more than twice as bad as being off by $25. MSE punishes large errors in this way

Train/Test Split

Before fitting the model, the dataset was split into training and testing data in the proportion of 70% training, 30% testing. This reserves a portion of the data to validate the train model on, to ensure that overfitting does not occur.

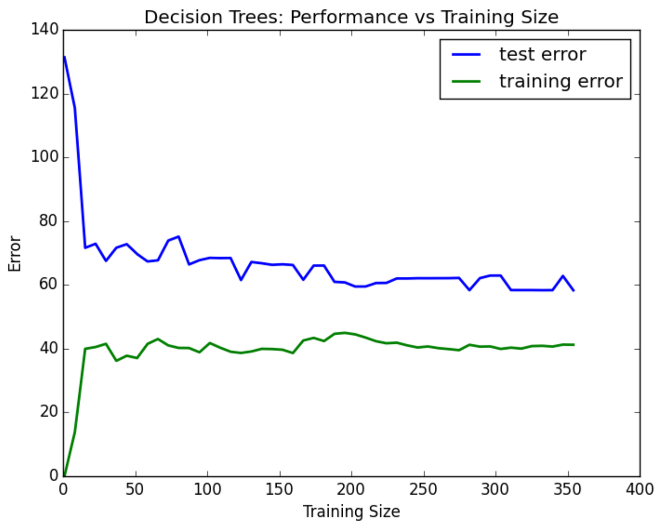
Analyzing Model Performance

Learning Curves

Using the split data, Decision Tree models were fit to the training and testing data. 10 different models were fit, each one with a different max depth parameter in the range from 1 to 10. Plots were made of each model's performance for the training and testing error vs the training size.

From these graphs it is clear the relationships between training size and training error, and training size and testing error. In general, training error increases (at a decreasing rate) as training size increases, and testing error decreases (also at a decreasing rate) as training size increases.

This produces plots like the one below, from the model fit with a max depth of 1:

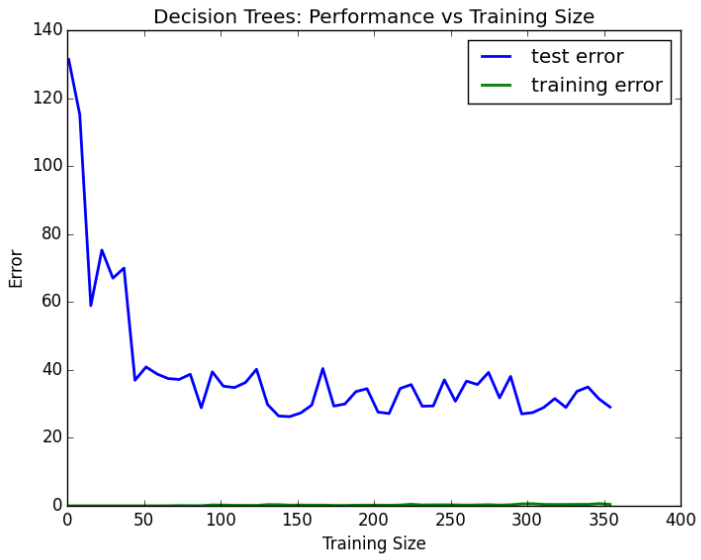


As you can see, initially the training error increases as the training size increases, but begins to level off after the training size is greater than about 50. This is due to the model overfitting the training examples when the number of training examples is small, but learning appropriate model parameters for the training data when there is more of it.

On the other hand, the testing error is high when the training size is small (due to the overfitting on the small number of training examples) and decreases as the training size increases. At about the same point that the training error leveled off, the testing error levels off as well.

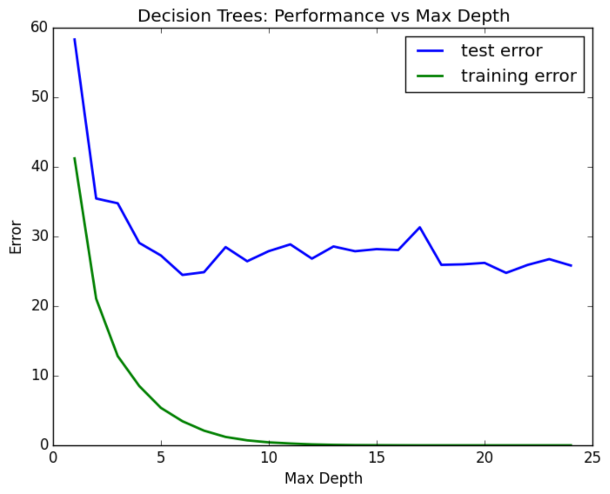
This model clearly underfits the data since it relatively high training and testing error compared to the other models, meaning it is not capturing the structure of the data well. This will be shown in subsequent graphs.

Here is the graph from the Decision Tree model with a max depth of 10:

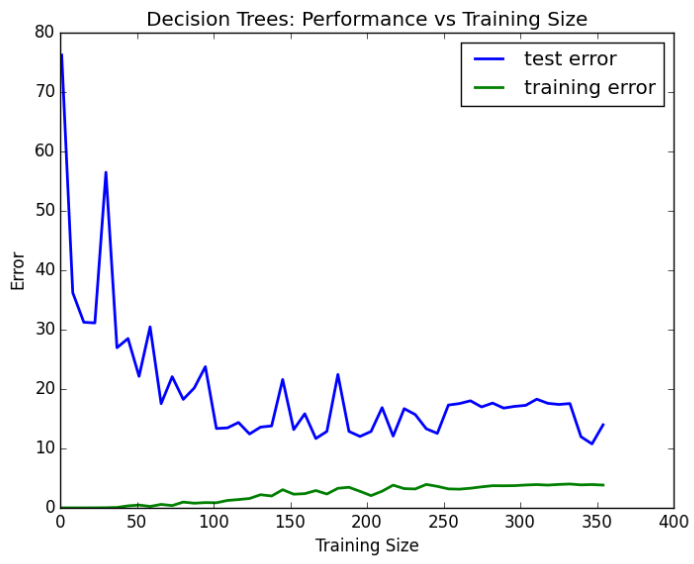


The larger max depth gives this model greater fitting ability, and it fits the training data nearly perfectly for all sizes of training data. Hence the training error is essentially 0 and barely visible on the graph. Similar to the model with a max depth of one, the testing error is initially high but decreases and then levels off as the training size increases. Even though this model is fitting the training data near perfectly, I would not say it is overfitting the data since its testing error is no higher than other models' testing error. Still, it is not the model I chose as the best.

Ultimately based on the learning curves, I ended up choosing the Decision Tree model with a max depth of 6. As you can see from the graph below, the testing error initially decreases as the max depth of the model increases. But after the max depth becomes larger than 6, the testing error remains relatively constant.



Since all else being equal a simple model is preferable to a more complex one, I chose the Decision Tree model with a max depth of 6 as the best. Here is a graph of its training and testing error:



As you can see, it's test error is relatively low and therefore the model should perform well even on new housing data points.

Grid Search

Scikit-Learn's powerful Gridsearch function was applied to the data to efficiently test Decision Tree models of differing max depths and predict with the best performing one. The default 3-fold cross validation was used, as well as the same MSE for the error function. It is important for grid search to use k-fold cross validation to ensure that the final model does not overfit on the training data and perform poorly on unseen examples.

Incredibly the training and evaluating of all of these models can be performed in one simple line of code:

reg = GridSearchCV(regressor, parameters, scoring=scorer)

And using the best_params_ function, it was found that the model with a max depth of 6 was the best performing. This is the same model I chose as optimal model from the learning curves section.

Model Prediction

The last part of the project was to use the optimal model as chosen from the grid search to determine a valid housing price based on an example with the following feature values:

```
House: [11.95, 0.0, 18.1, 0, 0.659, 5.609, 90.0, 1.385, 24,
680.0, 20.2, 332.09, 12.13]

Prediction: [ 20.76598639]

Best model parameter: {'max_depth': 6}
```

Based on these values, the model predicts a sale price very close to the training data median of 21.2. One way to test if this prediction seems reasonable, is to see if a majority of this training example's feature values are near their respective medians. An example with inputs near the median should produce an output near the median.

The following table shows the 25th, 50th, and 75th percentiles for the 13 features used to the train the model. The last row is "Yes" if the training example's value for the feature is between the 25th and 75th percentile.

	1	2	3	4	5	6	7	8	9	10	11	12	13
25th	.08	0.0	5.2	0.0	.45	5.9	45.0	2.1	4.0	279	17.4	375.3	6.9
50th	.26	0.0	9.7	0.0	.54	6.2	77.5	3.2	5.0	330	19.1	391.4	11.4
75th	3.6	12.5	18.1	0.0	.62	6.6	94.1	5.2	24	666	20.2	396.2	17.0
25-75	No	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	No	Yes

If you count them, that's 8 of the 13 feature values near the middle of their distribution. This serves as a good litmus test that the prediction of 20.766 is reasonable from the given inputs.